Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт космических и информационных технологий
Кафедра «Информатика»

УТВЕРЖДАЮ
Заведующий кафедрой
_____ Кузнецов А. С.
«___» _____ 2019 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Алгоритмическое и программное обеспечение оценки качества приложений, работающих в среде распределенных реестров

09.04.04 Программная инженерия
09.04.04.01 Программное обеспечение вычислительной техники и автоматизированных систем

Научный руководитель	_____	доцент, канд. техн. наук	А. С. Кузнецов
Выпускник	_____		Д. Б. Максимов
Рецензент	_____	доцент, канд. техн. наук	Д. В. Капулин

Красноярск 2019

РЕФЕРАТ

Выпускная квалификационная работа по теме «Алгоритмическое и программное обеспечение оценки качества приложений, работающих в среде распределенных реестров» содержит 52 страницы текстового документа, 26 использованных источников, 22 иллюстрации, 3 таблицы и 7 формул.

СРЕДА РАСПРЕДЕЛЕННЫХ РЕЕСТРОВ, УМНЫЕ КОНТРАКТЫ, ФОРМАЛЬНАЯ ВЕРИФИКАЦИЯ.

Объект исследования – формальная верификация.

Предмет исследования – формальная верификация программного обеспечения, работающего в среде распределенных реестров.

Цель: исследование применимости методов формальной верификации для анализа систем, работающих в среде распределенных реестров.

Задачи:

- провести анализ существующих программных решений для выполнения формальной верификации систем и выбрать подходящее решение для приложений, работающих в среде распределенных реестров;

- разработать модель и ограничения для системы работающей в среде распределенных реестров;

- провести исследование модели и проверить ограничения.

В результате была экспериментально проверена гипотеза о том, что возможно построение и формальное исследование модели умного контракта для регистрации новых пользователей и ее ограничений. Формальная верификация построенной модели позволила оценить степень достоверности сформулированного ограничения.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время, очень много задач решается посредством глобальной сети Интернет. В связи с ростом объема и важности хранимых и передаваемых данных возрастает уровень требований к безопасности системы в процессе проведения операций и хранения данных. Одним из возможных решений, которое позволяет повысить безопасность пользователей и их данных является концепция среды распределенных реестров. В связи с ростом популярности данной технологии, увеличивается количество продуктов, которые основываются на работе в среде распределенных реестров. Так как на данный момент большинство продуктов системы связаны с финансовыми операциями, участились случаи получения злоумышленниками средств пользователей системы. Данные инциденты в большинстве случаев происходят за счет ошибок в программном обеспечении, которых не удалось выявить на этапе тестирования. В данной работе показано как можно применить формальные методы для проверки разрабатываемых систем и их свойств.

Целью данной работы является исследование применимости методов формальной верификации для анализа систем, работающих в среде распределенных реестров.

Для достижения поставленной цели необходимо решить следующие задачи:

- провести анализ существующих программных решений для выполнения формальной верификации систем и выбрать подходящее решение для приложений, работающих в среде распределенных реестров;

- разработать алгоритмическую модель и ограничения для системы, работающей в среде распределенных реестров;

- провести исследование модели и проверить ограничения;

- сделать вывод о возможности применения методов формальной верификации для анализа систем, работающих в среде распределенных реестров.

Научная новизна заключается в проверке возможности использования методов формальной верификации для анализа программного обеспечения, работающего в среде распределенных реестров, так как на данный момент не было проведено исследований по успешному применению методов в данной области.

1 Анализ средств формальной верификации систем, работающих в среде распределенных реестров

1.1 Распределенный реестр

Распределенный реестр – это созданная по определенным правилам цепочка из блоков, записей, копии которой хранятся распределенно у других пользователей. При интеграции в реальную систему данная технология комбинируется вместе с другими технологиями криптографии, управлением данных и сетевого взаимодействия. Цепочки из блоков связываются с помощью хэш функции, которая вычисляет хэш сумму для блока, и зависит от значения хэш суммы предыдущего блока, что делает невозможным внедрение в цепочку сторонних блоков [1].

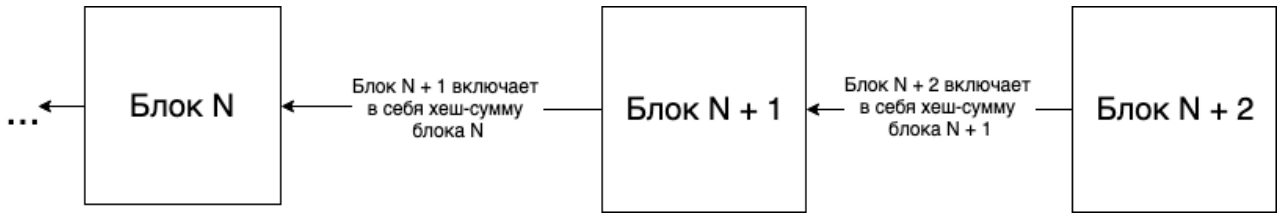


Рисунок 1 – Структура данных распределенного реестра

Структура данных распределенного реестра представлена на рисунке 1. Данная архитектура и позволяет безопасно выполнять, проводить их проверку и запись транзакций между пользователями [2].

Транзакция это операция, которая по завершению своей работы обновляет состояние среды распределенных реестров. Обычно транзакции содержат информацию по переводу средств между пользователями, но также могут включать в себя информацию из внешних систем. Жизненный цикл транзакции представлен на рисунке 2.

Рисунок 2 – Жизненный цикл транзакции

Можно выделить следующие этапы:

- создание транзакции – формирование сущности транзакции в системе;

- подпись ЭЦП – после создания транзакция обязательно должна быть подписана пользователем, который ее создал, что в свою очередь обеспечивает возможность тратить деньги, создавать умные контракты или передавать параметры связанные с транзакцией;

- валидация – подписанная транзакция отправляется на ноду - один из многих вычислительных центров распределенной сети, в котором проверяется валидность транзакции;

- распространение – если транзакция успешно прошла валидацию в ноде, то она распространяется по всем остальным нодам сети для проверки;

- после того как все вычислительные центры приходят к общему согласию что транзакция валидна – с этого момента она считается верифицированной и записывается в реестр;

- последним шагом в жизненном цикле транзакции является алгоритм подтверждения выполненной работы.

Первая работа по блокам, связанным криптографической защитой были описаны еще в 1991 году Стюартом Хобером и Вальтером Сторнеттой. Целью их работы было разработать систему, в которой метки времени, проставляемые документам не могли быть скомпрометированы [3]. Позже в 1992 году им удалось разработать концепцию дерева хэшей, которая позволила повысить эффективность работы системы хранения документов, за счет того, что они были преобразовались в один блок. Однако выработать концепт идеи распределенных реестров удалось Сатоши Накамото в 2008 году и уже в следующем году она была реализована в качестве компонента-криптовалюты «Биткоин» в которой центральное место занимает система доказательства правильности работы Hashcash. Однако как технология, которая была бы универсальной и позволила разработчикам применять данную концепцию в других сферах кроме финансовой, она сформировалась в 2013 году, когда появилась платформа Ethereum как альтернатива Биткоину. Успех данной платформы предопределило наличие скриптового языка программирования Solidity, который дал возможность разрабатывать приложения, работающие в среде распределенных реестров под различные сферы использования [4].

В настоящий момент концепция среды распределенных реестров помимо финансового сектора, в котором он активно применяется в криптовалютах,

используют во многих сферах жизни человека. Например, в медицине уже активно применяется программное обеспечение, которое позволяет работать с данными пациентов из любой точки мира, и не зависит от работы одного централизованного сервера. Также технология распределенных реестров уже применяется для контроля государственных границ Нидерландов. Данные и метрики пассажиров надежно хранятся в распределенной системе. Помимо государственного использования, технология работы в среде распределенных реестров используется крупными компаниями, такими как Amazon. Данная компания применяет преимущества технологии распределенных реестров для улучшения безопасности баз данных [5].

1.2 Умные контракты

Первые идеи умных контрактов были предложены в 1994 году Ником Сабо. Он описал умный контракт как компьютерный протокол, который на основе математических алгоритмов самостоятельно проводит сделки с полным контролем над их выполнением.

Впервые идеи Сабо воплотились на практике вместе с появлением первой криптовалюты биткоин и лежащей в ее основе технологии распределенных реестров. Некоторые принципы умных контрактов были заложены в протоколе биткоина. Однако большинство современных систем, работающих в среде распределенных реестров, включая биткоин, не обладают полнотой по Тьюрингу, поэтому их «контракты» представляют собой относительно простые конструкции, такие как мультиподпись или транзакции с отложенным исполнением.

Широкое практическое применение умные контракты получили с появлением и развитием проекта Ethereum. В 2013 году будущий его основатель Виталик Бутерин пришел к выводу, что биткоин плохо подходит в качестве базового протокола для умных контрактов, поскольку изначально не был

спроектирован под эту задачу. Впоследствии Бутерин решил создать с нуля наиболее подходящий для умных контрактов протокол [6].

Умный контракт может выполнять следующий функционал:

- регуляция споров между пользователями;
- предоставление своего функционала другим умным контрактам;
- хранение информации о приложении.

Потенциальные возможности и сферы использования умных контрактов обширны – от простой мультиподписи до операций с производными финансовыми инструментами. Мультиподпись – классический пример умного контракта. С ее помощью не доверяющие друг другу контрагенты могут заморозить некоторую сумму монет в среде распределенных таким образом, что в случае необходимости потратить эту сумму потребуются подписи более половины участников.

Умные контракты широко используются в сфере первичных распределений монет. Например, умный контракт может быть запрограммирован таким образом, что отправляя криптовалюту на кошелек проекта; если же финансовая цель будет достигнута, то средства будут перечислены разработчикам. Однако сделано это будет при условии, что достаточное число участников мультиподписи активируют свои ключи, тем самым лично подтвердив добросовестность проекта.

К наиболее перспективным сферам применения умных многие эксперты относят финансовый рынок (банковские услуги, страхование, торговлю деривативами), бухгалтерский учет и аудит, управление цепями поставок и логистику, регистрацию прав собственности, всевозможные голосования, умный транспорт, цифровую идентификацию личности и т. д.

Умные контракты существуют на тех же условиях в среде распределенных реестров что и настоящие пользователи, за исключением того, что они не могут самостоятельно инициировать выполнение, а только при помощи пользователей либо других умных контрактов [7].

Взаимодействие, которое происходит между пользователями или умными контрактами называется транзакциями. Так ресурсы среды распределенных реестров являются ограниченными каждая транзакция имеет свою стоимость. Это в свою очередь позволяет обеспечить безопасность системы от DoS-атак, когда происходит выполнение большого количества транзакций с целью довести работу системы до отказа [8].

Типичный сценарий взаимодействия между пользователями и умным контрактом представлен на рисунке 2.

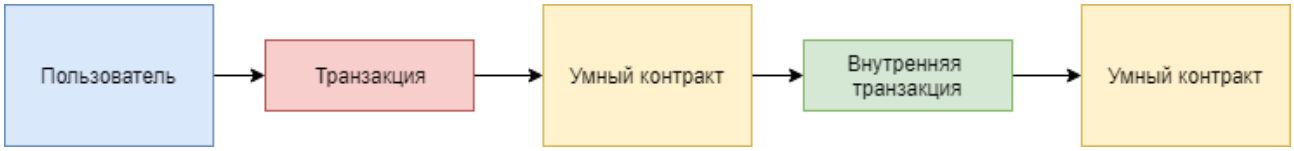


Рисунок 3 – Взаимодействие внутри среды распределенных реестров

Пользователь формирует транзакцию, которая инициирует работу умного контракта. В свою очередь умный контракт формирует внутреннюю транзакцию, которая в свою обращается к другому умному контракту. Таким образом, благодаря умным контрактам можно выстроить цепочку взаимодействий внутри среды распределенных реестров и обеспечить безопасность выполнения контракта между сторонами.

Однако для того чтобы полагаться на умный контракт и все его преимущества, необходимо быть уверенным что код, на котором базируется умный контракт, всегда выполняется надежно и даст гарантию того, что он будет работать в соответствии с ожидаемым поведением.

1.3 Формальная верификация

За прошедшее с момента появления технологии распределенных реестров время злоумышленниками было проведено множество попыток повлиять на работу приложений, основанных на данной технологии. Так как большинство

атакуемых систем были связаны с криптовалютами, целью данного вмешательства было вывести деньги пользователей за счет уязвимостей системы. В итоге за счет успешных хакерских атак злоумышленниками было выведено почти 2 миллиарда долларов [9].

Таблица 1 – Атаки на системы распределенных реестров

Название атаки	Причина атаки	Выведенная валюта, $
AllinVain	Злоумышленник проникал в жесткий диск жертв и переводил средства на внешний кошелек.	500.000
BitFloor	Злоумышленник заполучил незашифрованные ключи, которые хранились в сети для резервной копии.	85.000
Mt. Got 2nd Hack	Уязвимость при сохранении транзакций в среде распределенных реестров, позволяла злоумышленнику использовать цифровую подпись пользователя и изменить идентификатор транзакции.	350.000.000
The DAO	Уязвимость в коде позволяла вызываемым извне смарт-контрактам перехватывать поток управления и изменять данные пользователя.	55.000.000
Bitfinex	Злоумышленники нашли уязвимость в архитектуре кошельков с функцией мульти подписи.	72.000.000

В таблице 1 сведены самые известные случаи атак на приложения, работающие в среде распределенных реестров.

В большинстве случаев такие атаки оказывались успешными только потому что злоумышленники находили изъяны в функционировании

приложений, работающих в среде распределенных реестров. Из-за анонимности участников системы найти и наказать злоумышленников практически никогда не удается. Так как среда распределенных реестров является саморегулирующейся системой и нет возможности в случае критической ситуации вмешаться в ее работу, необходимо чтобы разрабатываемая система отвечала всем требованиям надежности.

Согласно одному из исследований, представленных на рисунке 4 основные ошибки при создании программного обеспечения возникают на этапе написания кода [10].

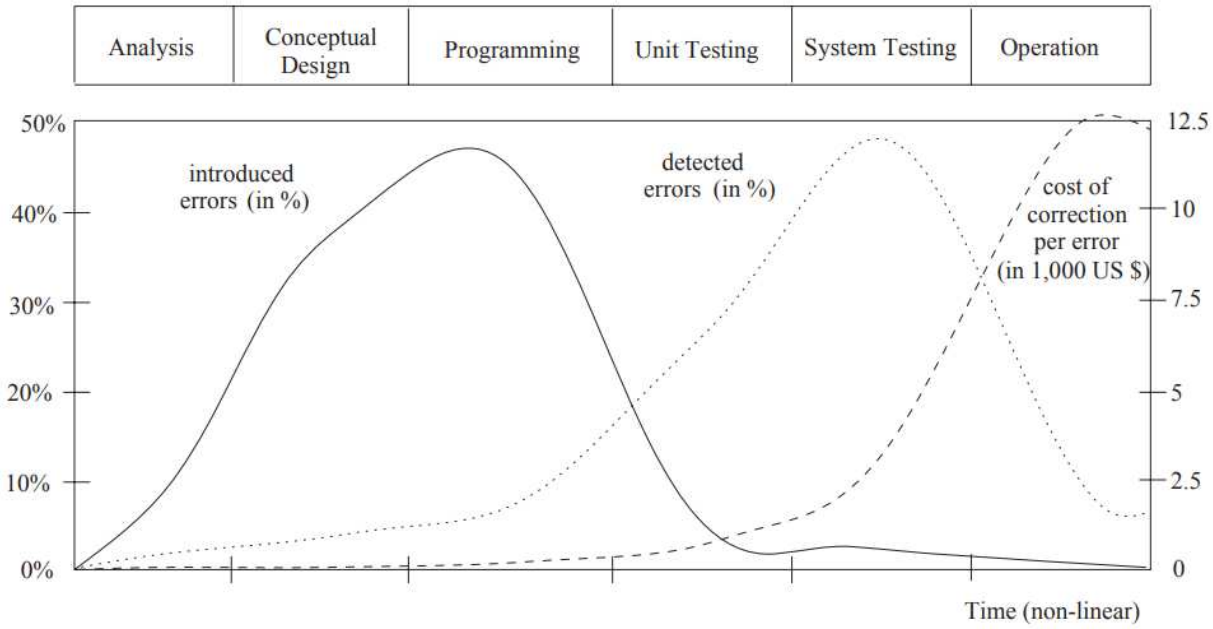


Рисунок 4 – Исследование распределения источников возникновения ошибок при разработке программного обеспечения

Около 15 % дефектов возникают в процессе проектирования программного обеспечения и большинство ошибок потом обнаруживается в процессе тестирования. Чем позже будет идентифицирована проблема, тем больше будет цена ошибки.

Для того чтобы проверить корректность и надежность выполнения программного обеспечения, применяют различные методы тестирования. Оно

помогает выявить отклонения работы программы от ожидаемого поведения для заранее прописанных сценариев – осуществляется проверка функциональных свойств системы. При тестировании программного обеспечения формируются входные данные, которые передаются системе, и проверяется работа системы на выходе. Однако, помимо функциональных свойств существуют еще свойства безопасности и защищенности системы, которые являются прямой противоположностью – если в первом случае проверяется – что система должна делать, то во втором уже необходимо наоборот найти уязвимые места в системе – то есть то, что система делать не должна. Но при проверке таких типов свойств почти невозможно знать сразу, какими уязвимостями может обладать проверяемая система. Для того чтобы найти эти уязвимости или наоборот убедиться что, система удовлетворяет свойствам безопасности и защищенности необходимо подвергнуть систему математическому анализу [11].

Для того чтобы формально доказать или, наоборот, опровергнуть корректность работы системы можно применить методы формальной верификации. Формальная верификация представляет собой набор инструментов, которые позволяют проанализировать всевозможный набор состояний разрабатываемой системы с целью выявления некорректного поведения [12].

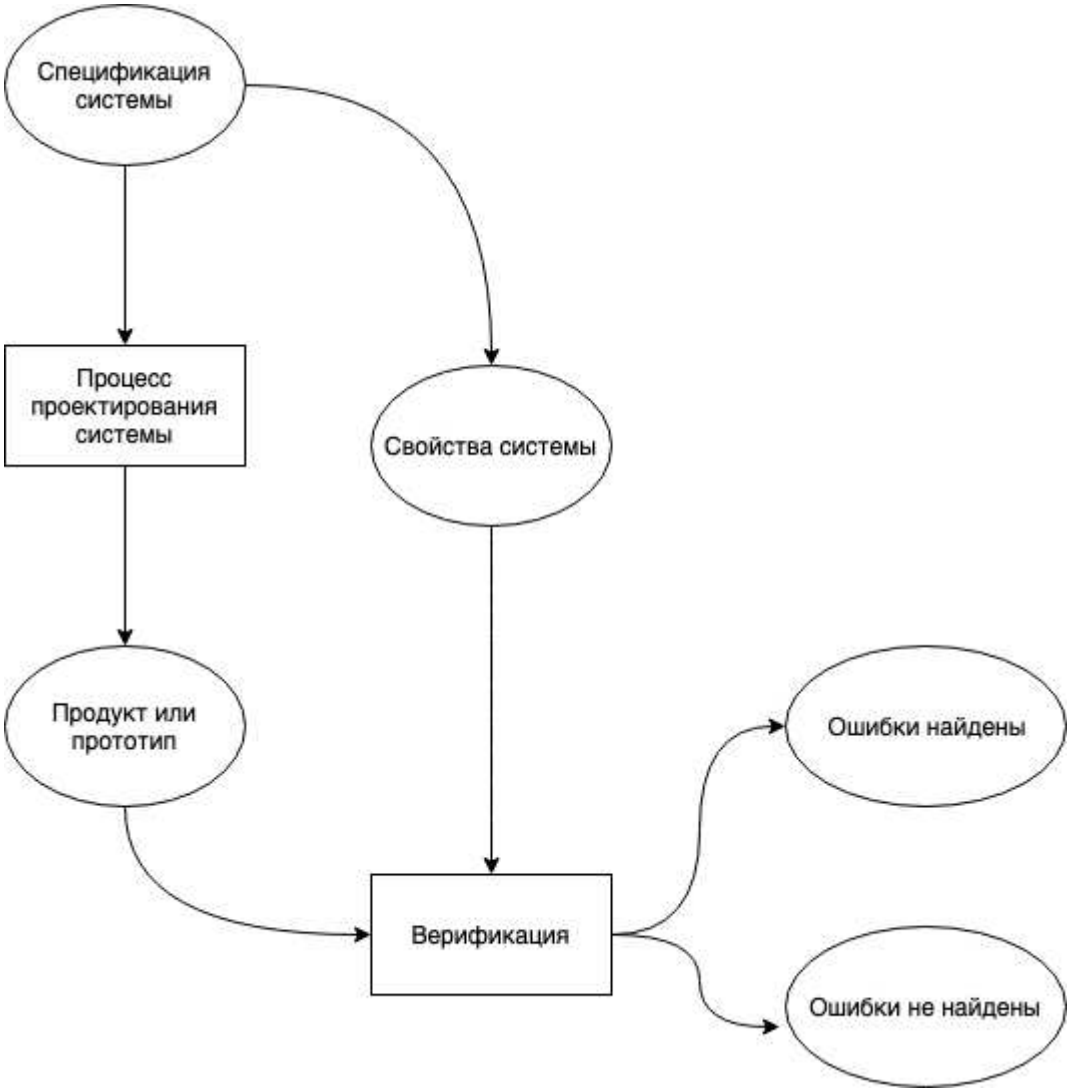


Рисунок 5 – Общая схема верификация системы

На рисунке 5 представлена обобщенная схема по верификации программного обеспечения. Одним из ключевых элементов при проведении верификации является спецификация системы. Спецификация описывает то, что система должна делать, а что не должна. Если в процессе верификации будет найден «дефект», то значит система, не соответствует одному из свойств спецификации. Система будет являться корректной только в том случае, если все условия спецификации будут выполнены. Поэтому понятие корректности в данном случае это не абсолютное свойство системы и полностью зависит от спецификации.

Методы формальной верификации можно разделить на две категории:

- методы проверки модели, которые проверяют структуру системы, которая задается в виде модели с помощью предметно-ориентированного языка программирования (DSL);

- методы доказательства теорем, сочетающие в себе автоматические техники для доказательства или опровержения различных свойств системы.

Граница между данными двумя категориями довольно размыта, и программные средства которые используются для верификации программного обеспечения могут одновременно использовать обе категории методов. Однако методы проверки моделей является более понятным разработчику, в то время как методы доказательства теорем требуют определенную математическую подготовку для работы [13].

Плюсами формальной верификации является достоверность и полнота получаемого результата, так как анализ, проводимый при помощи методов упомянутых выше охватывает все возможные состояния системы. Однако, затраты на формальную верификацию на порядок превышают затраты на тестирование, в связи с чем в большинстве случаев обходятся классическими методами тестирования программного обеспечения. Однако когда стоимость невыявленной вовремя ошибки в работе системы имеет большую стоимость, тогда методы формальной верификации являются более предпочтительным вариантом проверки системы. Учитывая опыт финансовых потерь при атаках злоумышленников на системы, работающих в среде распределенных реестров, использование методов формальной верификации при разработке умных контрактов поможет выявить критические места и повысить безопасность разрабатываемых систем [21].

1.4 Проводимые исследования

В настоящий момент проводится большое количество исследований по формальной верификации программного обеспечения.

При разработке космических шаттлов в NASA были проведены исследования группой ученых под руководством Стейси Нельсона и успешно применены методы формальной верификации в процессе разработки системы жизнеобеспечения летательных аппаратов [14]. Данная система позволяет обнаруживать, обрабатывать, а в отдельных случаях и заниматься исправлением ошибок, которые происходят во время полета корабля без вмешательства человека. Программное обеспечение, на котором был основан этот процесс, базируется на диагностической системе Livingstone, которая также была разработана в NASA. Данная система описывает нормальные и аномальные режимы для каждого компонента системы. Livingstone занимался мониторингом команд, поступающих на вход системе корабля и использовал модели чтобы предсказывать дальнейшее поведение системы. Для проведения формальной верификации были использованы следующие методы:

- проверка теорем;
- статический анализ кода;
- мониторинг в режиме реального времени.

Проводимое исследование показало, присутствие важных критических уязвимостей, связанных с отсутствием обновления данных в системе.

Также проводились исследования группой ученых из Шеффилдского университета возглавляемых профессором Сандором Вересом, в котором ставились задачи проверки возможности возвращения управления квадрокоптером пилоту, вследствие различных внештатных ситуаций [15]. В процессе исследования была изучена работа системы в динамике и был проведен анализ устойчивости квадрокоптера при полете. Благодаря этому и была составлена модель системы. В итоге, по данной модели была проведена формальная верификация системы, целью которой было проверить корректную работу модуля управления в заданных условиях.

В одном из исследований Гарвардского университета под руководством Энтони Делигната было предложено помимо верификации, основанной на

модели системы проводить верификацию исполняемого кода уже на уровне трансляции в машинный код, что позволило увеличить эффективность процесса формальной верификации [17].

1.5 Существующие решения

1.5.1 PRISM

PRISM – программное обеспечения для которое используется для вероятностной проверки моделей. Данное средство позволяет строить формальные модели систем, поведение которых носит случайный или вероятностный характер, и анализировать их. Это позволяет применять данное решение в различных областях, например в исследовании протоколов связи Bluetooth и FireWire, протоколов безопасности, распределенные алгоритмы и много другое [16]. Модели, которые строятся для исследований, описывают при помощи языка состояний PRISM. Данное программное обеспечение позволяет легко проводить количественный анализ свойств систем. Язык который используется для задания спецификации свойств системы, включает в себя языки темпоральной логики, такие как PCTL, CSL и LTL.

Данное программное обеспечение предоставляет все возможности для проведения анализа систем, работающих в среде распределенных реестров, однако в данной программе не предусмотрено генерация контр-примеров, которые могли бы помочь однозначно выявить недостатки системы и условия, при которых данные недостатки проявляются.

1.5.2 Isabelle

Isabelle – это средство для доказательства теорем общего назначения. Isabelle был разработан совместно Кембриджским и Мюнхенскими

университетами, но на данный момент уже большое количество как институтов так и отдельных людей по всему миру внесли свой вклад в развитие данное программного обеспечения [18]. Он позволяет выражать математические формулы через формальный язык, и предоставляет инструменты для доказательства этих формул с помощью логического исчисления. При формализации математических выражений используется логика высшего порядка, которую можно использовать для описания больших приложений. Доказательства проводятся при помощи структурированного языка доказательств Isar, который делает понятным текст доказательства как для людей, так и для вычислительной машины. Одним из преимуществ данного продукта, является возможность трансляции исполняемых спецификаций в код на таких языках программирования как SML, OCaml, Haskell и Scala что позволяет выполнять проверку в привычной программистам среде разработки.

Isabelle очень удобен для доказательства различных математических теорем, свойств элементарных функций и нахождении контрпримеров для опровержения проверяемых условий, однако данные особенности налагают ограничения на область применения данного продукта и используются больше для научных исследований нежели для прикладных программ и не позволяют исследовать работу и свойства системы в динамике.

1.5.3 K-Tool

K-Tool – это фреймворк, который основывается на заранее составленной семантике языка программирования, благодаря которой возможно использовать рассматриваемой программу, в качестве конфигурации для последующего анализа [19].

K-Tool был разработан совместными усилиями разработчиков из США и Румынии.

Данное программное обеспечение, основываясь на семантике языка программирования, на котором написана анализируемая программа, строит цепочки – последовательности из всех возможных состояний системы, что и позволяет проводить формальную верификацию систем, выполнять проверку моделей, запускать выполнение программ, при помощи интерпретаторов и интерактивно воздействовать на них.

Для того, чтобы работать с данным фреймворком, необходимо иметь правильно составленную семантику языка анализируемой программы. Не смотря, на то что семантики для популярных языков программирования таких как C, Java, JavaScript активно развиваются благодаря открытым исходным кодам, анализ в области распределенных реестров и в частности умных контрактов требует своей семантики, построенной на языке программирования Solidity. Однако составление семантики языка является трудозатратным и долгим процессом.

1.5.4 BIP Framework

BIP Framework – это инструмент для построения моделей систем, которые будут являться разновидностью сетей Петри, которая расширяется за счет введение переменных и функций, написанных на языке программирования C [20].

Особенностью данного фреймворка являются разделение работы модели на слои [23]:

- слой поведения – регулирует поведение внутри элементов системы – атомов;

- слой взаимодействия – ответственен за взаимодействие между атомами или совокупностями атомов при работе системы;

- слой приоритета – регулирует очередность между переходами состояния системы.

Данное разбиение на уровни функционирования модели, обеспечивает гибкость при построении самой модели а также предоставляет возможность описывать модель в декларативной форме, которая более привычна обычному разработчику.

Для того, чтобы исследовать свойства и ограничения моделируемой системы, необходимо формализовать их в виде формул линейной темпоральной логики.

BIP Framework использует модуль статистической проверки модели для запуска и проведения симуляции работы системы в процессе которого фиксируется данные по каждому состоянию системы и в итоге формируются трассы работы системы, на основе которых можно проводить проверку свойств и ограничений системы с помощью двух методов: проверки гипотез либо оценки вероятности. Также в системе можно проводить параметрические исследования – это автоматизированный способ выполнить проверку моделей на семействе свойств, которые выражены в виде функции $\phi(x)$, где x – это целочисленный параметр, для которого задается начальное значение, конечное значение и шаг изменения [20].

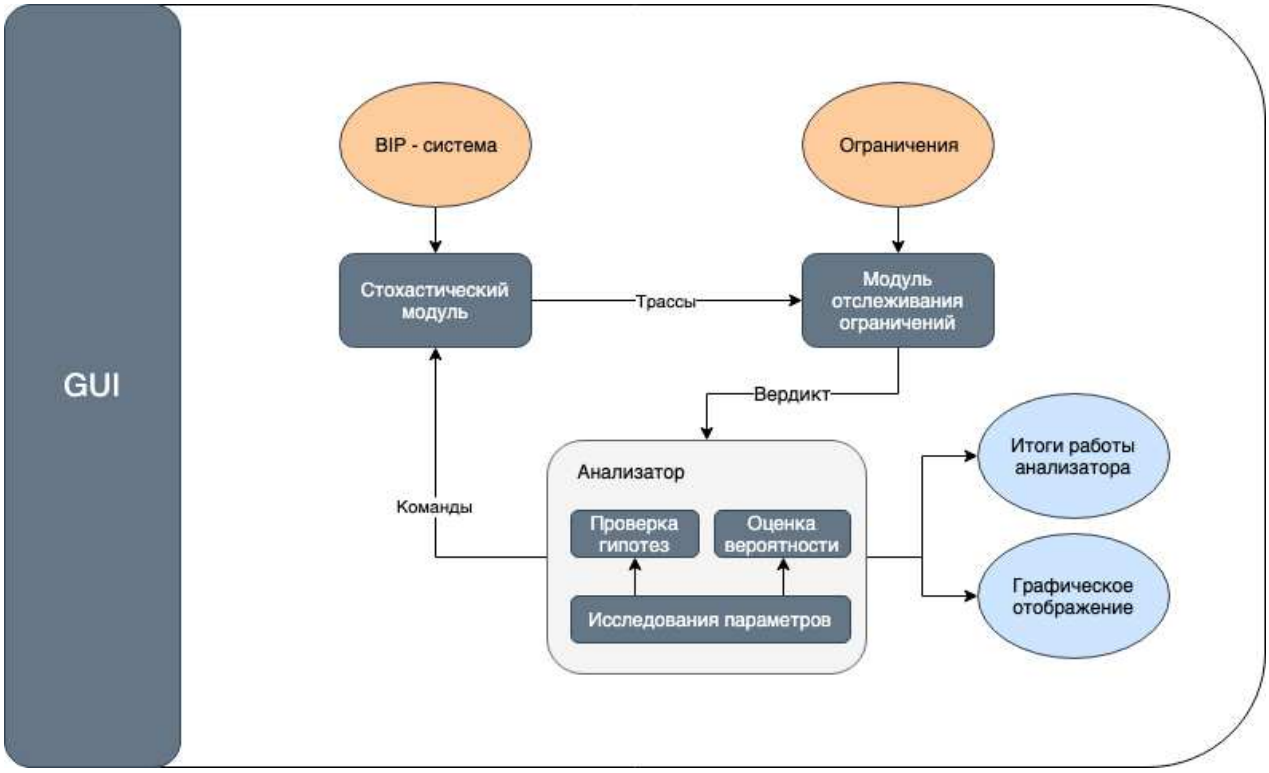


Рисунок 6 – Структура BIP Framework

Структура BIP Framework представлена на рисунке 6. Стохастический модуль запускает работу BIP – системы с параметрами исследования, которые поступают от анализатора. Результаты работы системы в виде трасс поступают на модуль отслеживания ограничений, который проверяет соблюдение ограничений во время работы системы. После этого данный модуль выносит вердикт. Все данные по исследованию собираются анализатором и выдаются пользователю по окончанию работы. Также анализатор формирует графическое отображение результатов в виде сводных графиков. Весь функционал приведенный выше доступен пользователю через GUI.

1.5.5 Выбор средства для проведения исследования

Для того чтобы выбрать средство для проведения исследования проведем сравнительный анализ между средствами, описанными выше.

Сравнительный анализ проводится по следующим критериям:

- моделирование стохастической системы;
- проведение параметрического исследования;
- формирование графиков сравнительного анализа;
- подключение пользовательского исполняемого кода.

Результаты сравнительного анализа представлены в таблице 2.

Таблица 2 – Сравнительный анализ программных средств

Программное средство	Критерии сравнения			
	Моделирование стохастической системы	Проведение параметрического исследования	Формирование графиков сравнительного анализа	Подключение пользовательского исполняемого кода
PRISM	+	+	+	-
K-tool	+	-	-	+
Isabelle	-	-	-	+
BIP Framework	+	+	+	+

По результатам анализа можно сделать вывод что инструмент BIP – Framework наилучшим образом подходит для выполнения данного исследования.

2 Конструирование модели исследуемой программной системы

2.1 Метод проверки моделей

В данной работе для исследования был выбран метод формальной верификации – проверка модели.

Метод проверки моделей – это автоматизированная техника, в которой исследуемая система представлена в виде конечного автомата. Целью является установление степени соответствия модели набору формальных ограничений. Языком описания ограничений является язык линейной темпоральной логики.

Распространенной практикой является автоматическая генерация модели системы по соответствующей формальной спецификации.

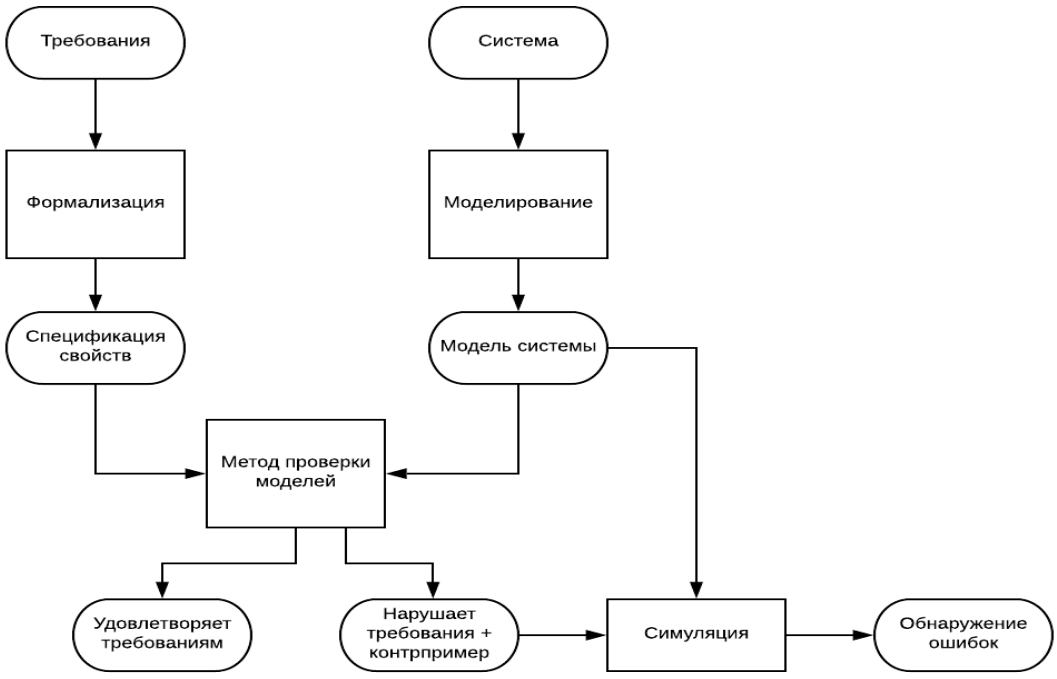


Рисунок 7 – Схема работы метода проверки моделей

Схема работы метода проверки моделей представлена на рисунке 7. Она отображает процесс верификации, который состоит из трех этапов:

- этап моделирования;

- этап симуляции;
- этап анализа.

2.1.1 Этап моделирования

На данном этапе происходит построение модели исследуемой системы. Модель системы должна описывать поведение системы в однозначном виде. Обычно модель представляет в виде конечного автомата, которая включает в себя набор состояний и набор переходов. Состояния содержат информацию о текущих значениях переменных. Переходы описывают то как система переходит из одного состояния в другое. Для описания обычно используется специальный предметно-ориентированный язык. Помимо модели на данном этапе необходимо задать ограничения для исследуемой системы. Обычно для этих целей используют языки темпоральной логики. Темпоральная логика является расширенной версией обычной логики с операторами, значение которых зависит от времени [26]. Это позволяет расширить спецификацию проверяемых свойств системы зачет проверки свойств корректности, достижимости, безопасности и свойств реального времени.

2.1.2 Этап симуляции

На этапе симуляции происходит систематический процесс построения трасс изменения состояния системы. В процессе построения происходит проверка всех возможных состояний системы.

2.1.3 Этап анализа

На данном этапе происходит оценка степени соответствия модели набору ограничений. Если модель удовлетворяет заданному ограничению, то

происходит переход на проверку следующего ограничения. В отдельных случаях бывает, что у программы заканчиваются ресурсы и происходить утечка памяти, в этом случае необходимо попытаться сократить саму модель, и повторить исследование. Если модель не удовлетворяет заданному ограничению то это может возникать из-за следующих причин:

- ошибка проектирования модели;
- ошибка проектирования системы;
- ошибка при задании ограничения.

2.2 Описание модели

В данном исследовании был смоделирован процесс регистрации пользователя в системе, а именно получение псевдонима – имени, которое предоставляется пользователю и связывается с его уникальным адресом в системе. Однако, количество псевдонимов, которое может выдать система, ограничено самим хранилищем псевдонимов и, учитывая скорость взаимодействия в среде распределенных реестров и то, что каждое действие при обмене данных в системе имеет свою стоимость, налагает ограничения на возможность пользователя сразу получить свой псевдоним, и ему придется предпринять больше одной попытки для его получения. После определенного количества попыток, если пользователь не смог получить псевдоним, система отправляет пользователю отказ в регистрации. Блок-схема общего алгоритма работы представлена на рисунке 8.

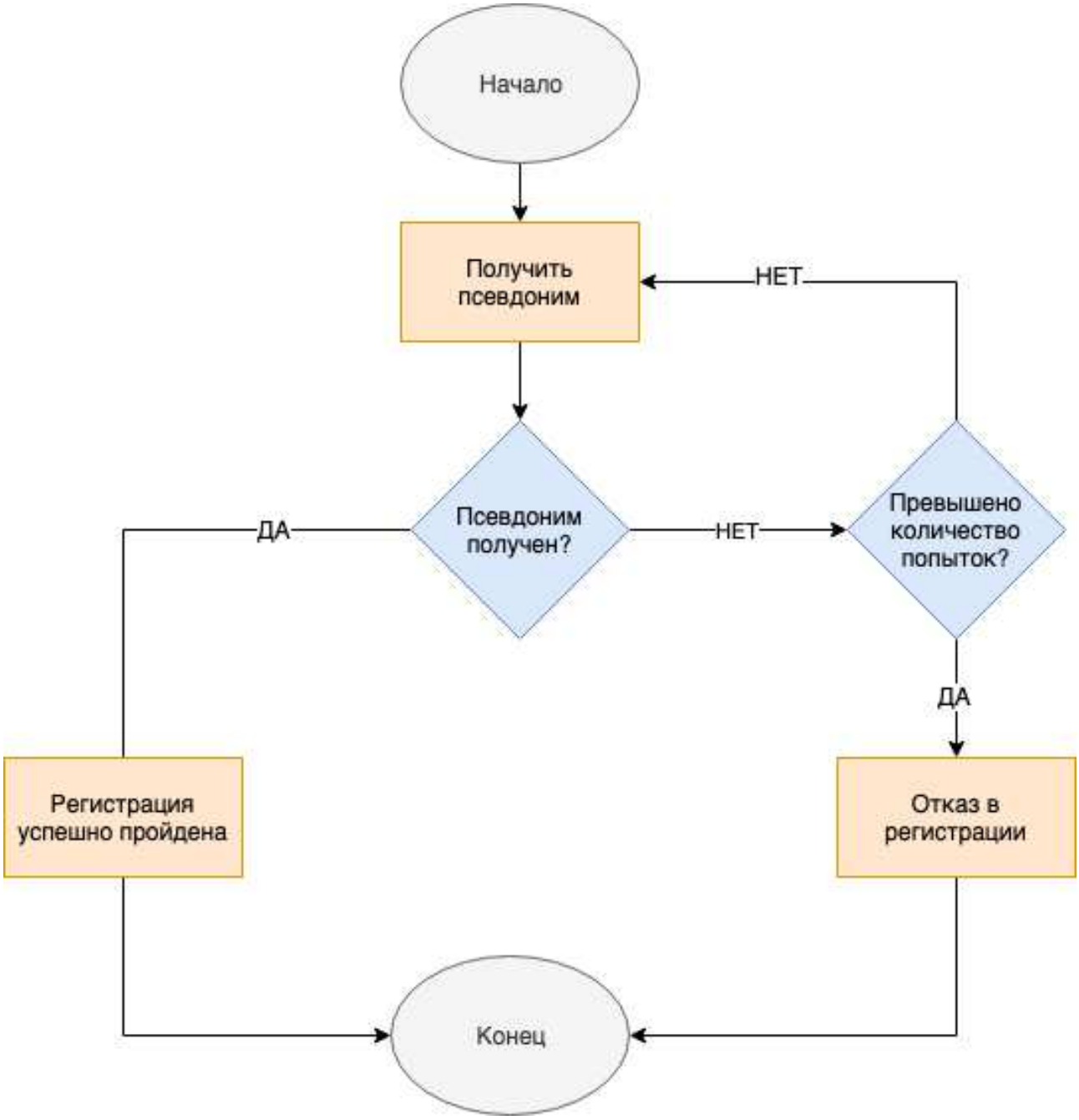


Рисунок 8 – Алгоритм регистрации пользователя

По представленному выше алгоритму строится модель системы в BIP формате.

Модель системы в BIP – формате в общем состоит из следующих элементов [8]:

- соединение – общая сущность в которой объединяются все компоненты системы;

- атомы – основные элементы системы, которые определены как самостоятельные элементы и не зависимые от других атомов системы. Каждый атом представлен в виде конечного автомата;

- коннекторы – элементы, используемые для обеспечения взаимодействия между атомами системы;

- порты – интерфейс для взаимодействия с атомом из внешней среды.

Схема для моделируемой системы получения пользователем псевдонима представлена на рисунке 9:

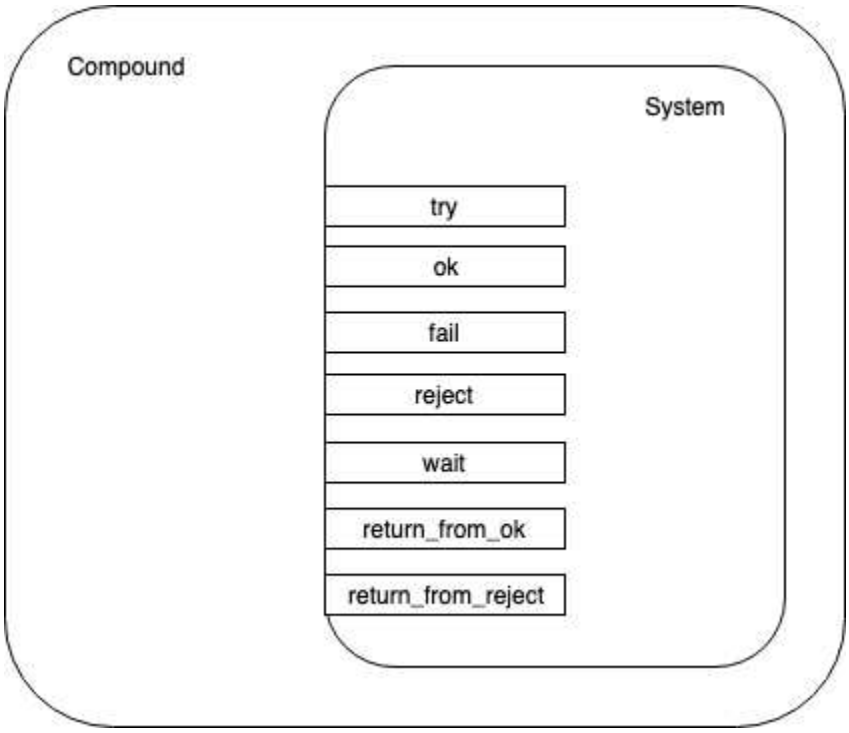


Рисунок 9 – Общая архитектура модели в BIP формате

При построении атома System необходимо определить следующие элементы:

- состояния системы представляют собой состояния конечного автомата;

- переходы между состояниями во время которых происходит выполнение различных вычислений;

- ограничения на переходы, если в определенный момент времени из состояния доступен переход сразу в несколько состояний, ограничения позволяют регулировать работу модели.

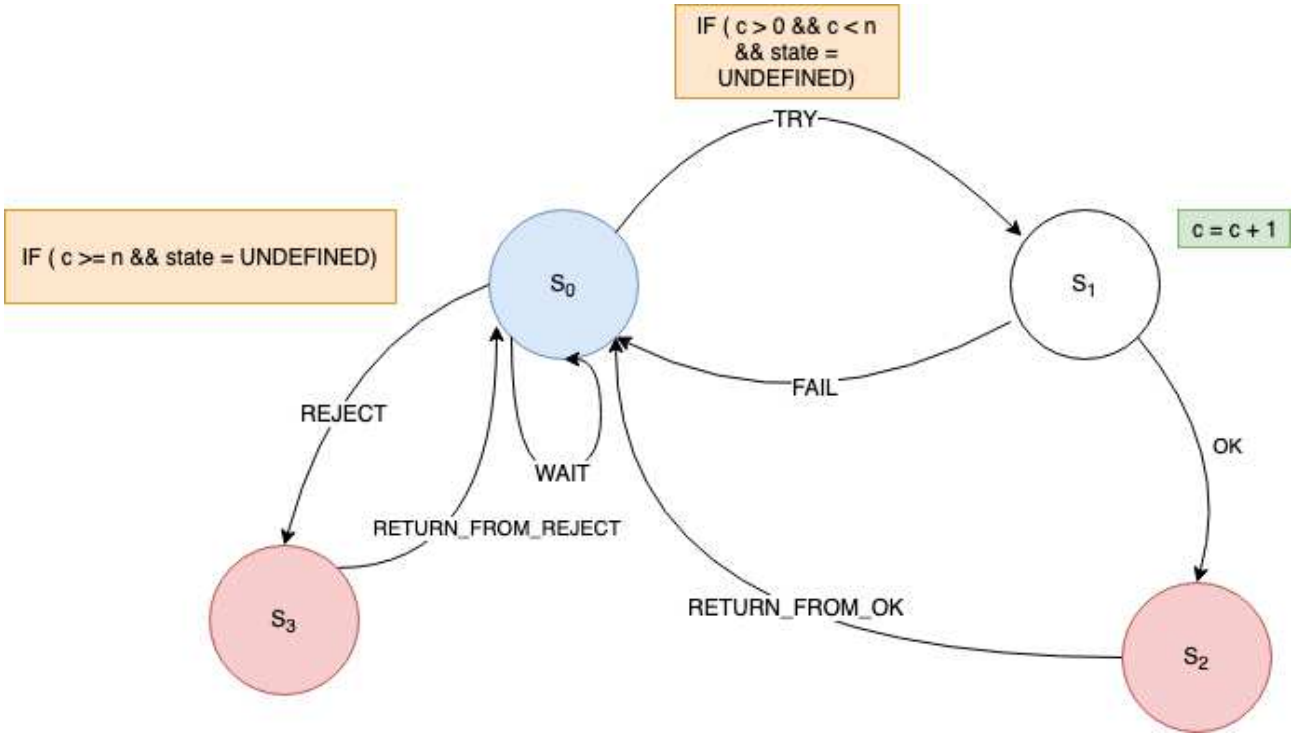


Рисунок 10 – Схема атома System

В схеме моделируемого атома, представленной на рисунке 10, показаны все состояния и возможные переходы между ними:

- s_0 – начальное состояние;

- s_1 – попытка зарегистрировать имя пользователем;

- s_2 – успешная регистрация и получение алиаса пользователем;

- s_3 – отказ в регистрации пользователю.

Состояния s_2 и s_3 являются конечными и служат индикаторами успешного завершения работы системы. Переходы между состояниями могут быть ограничены условиями.

Изменения между состояниями описаны следующими переходами:

- TRY – начало процесса регистрации, выполняется проверка количества попыток получить псевдоним из хранилища;

- OK – пользователь успешно прошел процесс регистрации;

- FAIL – хранилище не смогло выдать пользователю псевдоним, возвращение в начальное состояние;

- REJECT – после определенного количества попыток получить псевдоним пользователю придет отказ системы в получении псевдонима;

- RETURN_FROM_OK, RETURN_FROM_REJECT, WAIT – переходы, необходимые для корректного завершения исследования, служат для удержания системы в начальном состоянии.

2.3 Исследуемые ограничения

Основным требованием к описанной в данной работе системе является следующее: во время регистрации пользователь может получить псевдоним или отказ за количество попыток, не превышающее значение x. Данное требование представлено в виде формулы линейной темпоральной логики следующим образом:

$$\phi(t) \equiv F_{[0,t]}[\neg\,(System.state = 0) \wedge (System.c \leq x)], \quad (1)$$

где $\phi(t)$ – функция исследуемого параметрического требования;

$F_{[0,t]}$ –унарный оператор темпоральной логики Future в интервале от 0 до t;

System.state – состояние системы в текущий момент времени;

System.c – количество попыток, предпринятых пользователем для получения псевдонима;

x – значение исследуемого параметра.

В данном выражении System.state описывает состояние системы в текущий момент времени, где значение равное нулю – это неопределенное состояние

системы. System.c отражает количество попыток, предпринятых пользователем для получения псевдонима.

Помимо основного исследования проводятся проверка адекватности модели в условиях следующих ограничений:

- в определенный момент времени система должна перейти в конечное состояние успешной регистрации либо отказа в получении псевдонима, описываемое формулой 2.

$$\phi(t) \equiv F_{[0,t]}[\neg\,(System.state = 0) \wedge (System.c > 0)], \quad (2)$$

где $\phi(t)$– то же, что и в формуле (1);

$F_{[0,t]}$– то же, что и в формуле (1);

System.state – то же, что и в формуле (1);

System.c – то же, что и в формуле (1).

- система не должна сразу перейти в состояние отказа пользователю в регистрации и данное ограничение описывается формулой

$$\phi(t) \equiv N_{[0,t]}[\neg\,(System.state = 0)], \quad (3)$$

где $\phi(t)$– то же, что и в формуле (1);

$N_{[0,t]}$ – унарный оператор темпоральной логики Next;

System.state – то же, что и в формуле (1);

- на протяжении всей работы системы количество попыток по регистрации должно быть в границах от 1 до N, что отображено в следующей формуле

$$\phi(t) \equiv G_{[0,t]}[\,(System.state \geq 0)\ \&\&\ (System.c \leq System.N)], \quad (4)$$

где $\phi(t)$ – то же, что и в формуле (1);

$G_{[0,t]}$ – унарный оператор темпоральной логики Globally;

System.state – то же, что и в формуле (1);

System.c – то же, что и в формуле (1);

System.N - максимальное количество попыток для пользователя пройти регистрацию.

- на протяжении всей работы системы, количество попыток на текущем и предыдущем шаге должно всегда быть в интервале от 0 до 1, что выражено через следующую формулу

$$\phi(t) \equiv \phi_1(t) \;\&\&\; (System.c - System.\text{c\_}meta \leq 1)] \quad (5)$$

где $\phi(t)$ – то же, что и в формуле (1);

$\phi_1(t)$ – это формула (6);

System.c – то же, что и в формуле (1);

System.c_meta - значение параметра System.c на предыдущем шаге.

$$\phi(t) \equiv G_{[0,t]}[\,(System.c - System.\text{c\_}meta \geq 0\,) \quad (6)$$

где $\phi(t)$ – то же, что и в формуле (1);

$G_{[0,t]}$ – то же, что и в формуле (4);

System.c – то же, что и в формуле (1);

System.c_meta - то же, что и в формуле (5).

Также проведем дополнительное параметрическое исследование зависимости оценки вероятности завершить процесс регистрации от времени выполнения при помощи следующей формулы

$$\phi(t) \equiv F_{[0,x]}(!\,(System.state = 0)) \quad (7)$$

где $\phi(t)$– то же, что и в формуле (1);

$F_{[0,t]}$– то же, что и в формуле (1);

System.state – то же, что и в формуле (1).

2.4 Методы оценки результатов работы

Для оценки результатов работы метода проверки моделей, использован алгоритм статистической проверки для верификации стохастических систем – метод PESTIM [22]. В результате проверки заданных свойств системы данный метод должен дать на количественный вопрос: какова вероятность, что система S удовлетворяет заданному ограничению ϕ?

Настраиваемыми параметрами алгоритма PESTIM будут являться δ – точность, которая показывает, что процедура вычислит значение для p' такое, что $|p' - p| \leq \delta$ с доверительным интервалом $1 - \alpha$. Данный метод основан на границах Чернова-Хефдинга [24]. Практическое значение данных параметров заключается в том, что чем меньше значения α и δ, тем больше измерений необходимо провести для формирования вывода. В таблице 3 приведены значения зависимости количества измерений от значений α и δ.

Таблица 3 - Зависимость количества измерений от значений α и δ

Значение α	Значение δ	Количество измерений
0	0	0
0.1	0.1	1119
0.2	0.2	231
0.3	0.3	85
1	0.9	4

По приведенной выше таблице можно сделать вывод что, чем меньшее значение параметров α и δ будет выбрано, тем точнее будет проведен эксперимент.

Параметрическое исследование – это автоматизированный способ выполнить проверку моделей на семействе свойств, которые выражены в виде функции $\phi(x)$, где x это целочисленный параметр, для которого задается начальное значение, конечное значение и шаг изменения. Алгоритм параметрического исследования представлен на рисунке 11. При анализе входными данными будет являться система *S*, исследуемое параметрическое свойство $\phi(x)$и область определения *П.* Далее для каждого исследуемого значения *v* проводится симуляция работы системы и формируется трассы выполнения моделей. После чего проводится вынос вердикта относительно заданных ограничений и модуль формирует итоговое решение.

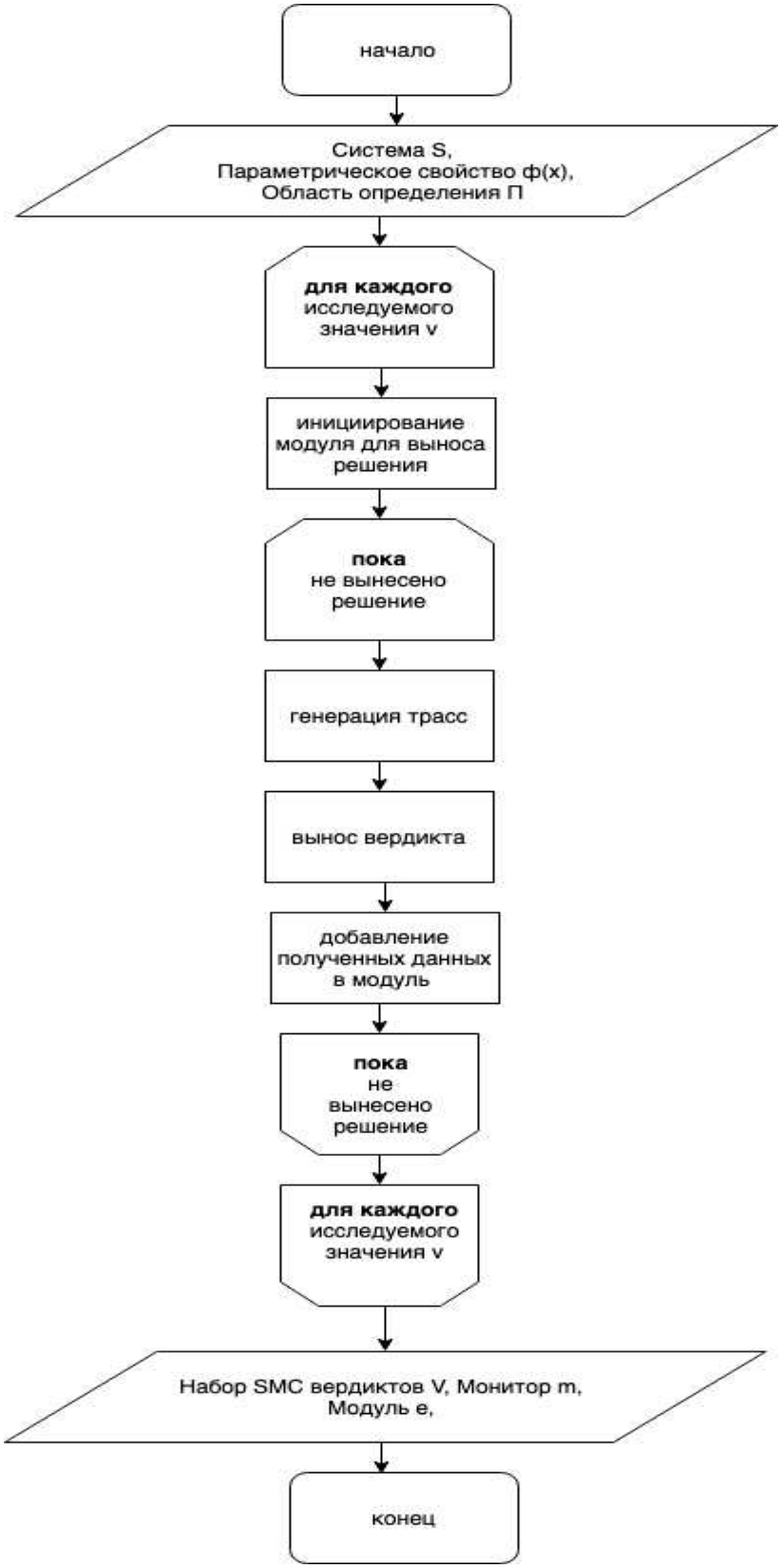


Рисунок 11 – Алгоритм параметрического исследования

3 Проведение исследования

3.1 Используемые модули для исследования

При подготовке к работе программного обеспечения BIP Framework необходимо обеспечить взаимодействие между модулями для проведения исследования. Для данной работы были выбраны следующие модули:

- модуль симуляции работы стохастических систем (SSE);
- модуль мониторинга (MM);
- модуль статистической проверки моделей (SMCE);
- модуль параметрического исследования (PEM).

3.2 Исследование процесса регистрации пользователя

Для того чтобы провести исследование процесса регистрации пользователя запустим и проанализируем работу системы с помощью ограничения, описанного формулой (1) при различном значении оценки вероятности получить пользователю свой псевдоним p'равному 0.1, 0.3, 0.5, 0.7 и 0.9.

Параметры с которыми, проводилось исследование:

- α= 0.1;
- β= 0.1;
- N = 5000;
- t = 10000.

На рисунке 12 представлено исследование с оценкой вероятности $p' = 0.1$.

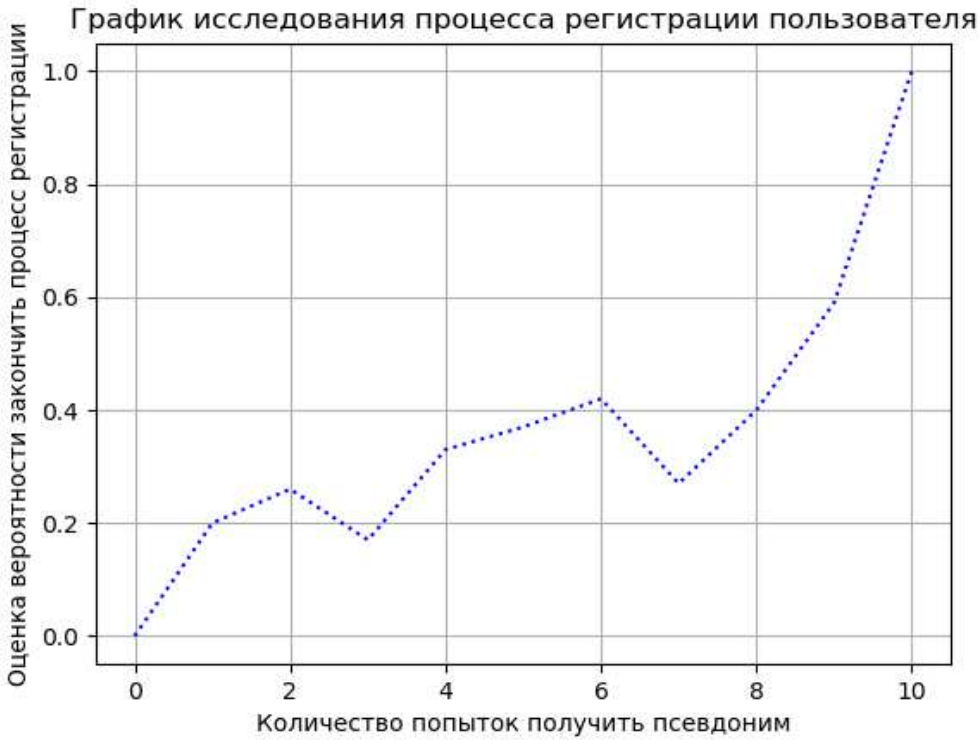


Рисунок 12– График исследования процесса регистрации пользователя при $p' = 0.1$

На рисунке 13 представлено исследование с оценкой вероятности $p' = 0.3$.

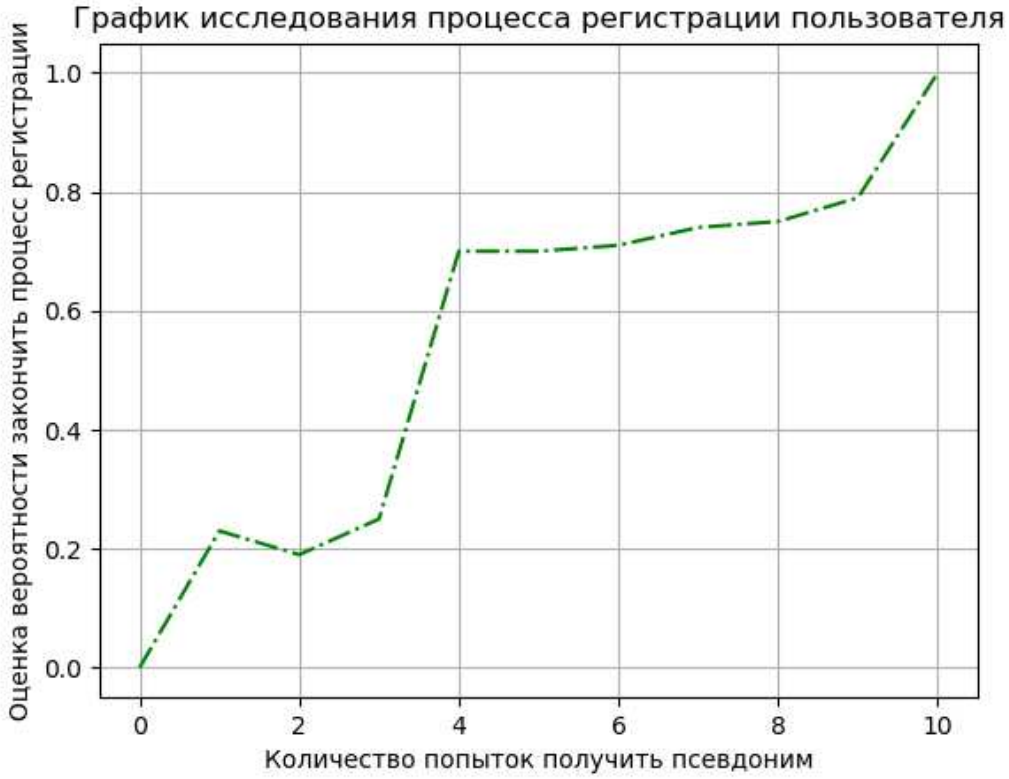


Рисунок 13 – График исследования процесса регистрации пользователя при $p' = 0.3$

На рисунке 14 представлено исследование с оценкой вероятности $p' = 0.5$.

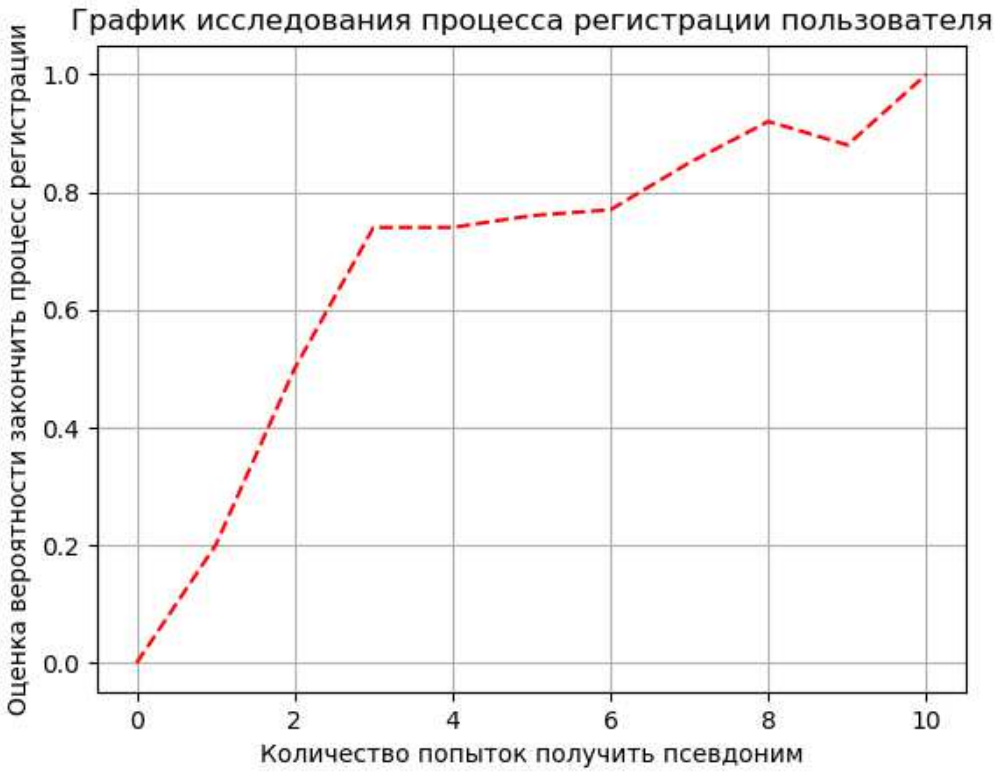


Рисунок 14 – График исследования процесса регистрации пользователя при $p' = 0.5$

На рисунке 15 представлено исследование с оценкой вероятности $p' = 0.7$.

Рисунок 15 – График исследования процесса регистрации пользователя при $p' = 0.7$

На рисунке 16 представлено исследование с оценкой вероятности $p' = 0.9$.

Рисунок 16 – График исследования процесса регистрации пользователя при $p' = 0.9$

По итогам пяти исследований сделаем сводный график для проведения сравнительного анализа результатов исследования. Данный график представлен на рисунке 17.

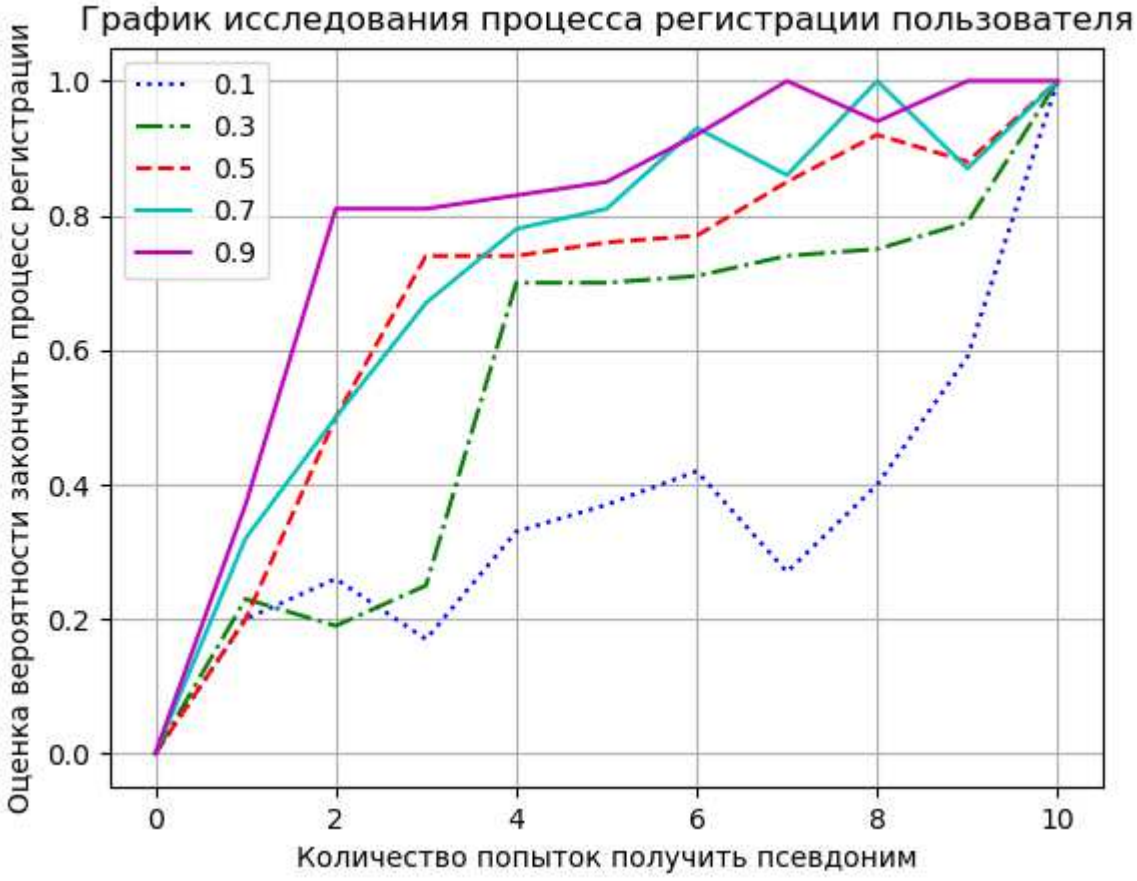


Рисунок 17 – Сводный график сравнительного анализа

По результатам данной серии исследований можно сделать вывод, что чем больше заполнено хранилище псевдонимов тем больше времени будет необходимо пользователю для того чтобы завершить процесс регистрации. Это в свою очередь потребует от пользователя большее количество ресурсов для получения псевдонима.

3.3 Проверка адекватности модели

Также была проведена проверка адекватности модели посредством формул 2,3,4,5,6,7 указанных в подпункте 2.3.

Исследования проводилось со следующими параметрами:

- α= 0.3;

- β= 0.3;

- N = 1000;

- t = 10000.

Результаты проверки ограничений по формуле (2) отображены на рисунке 18.

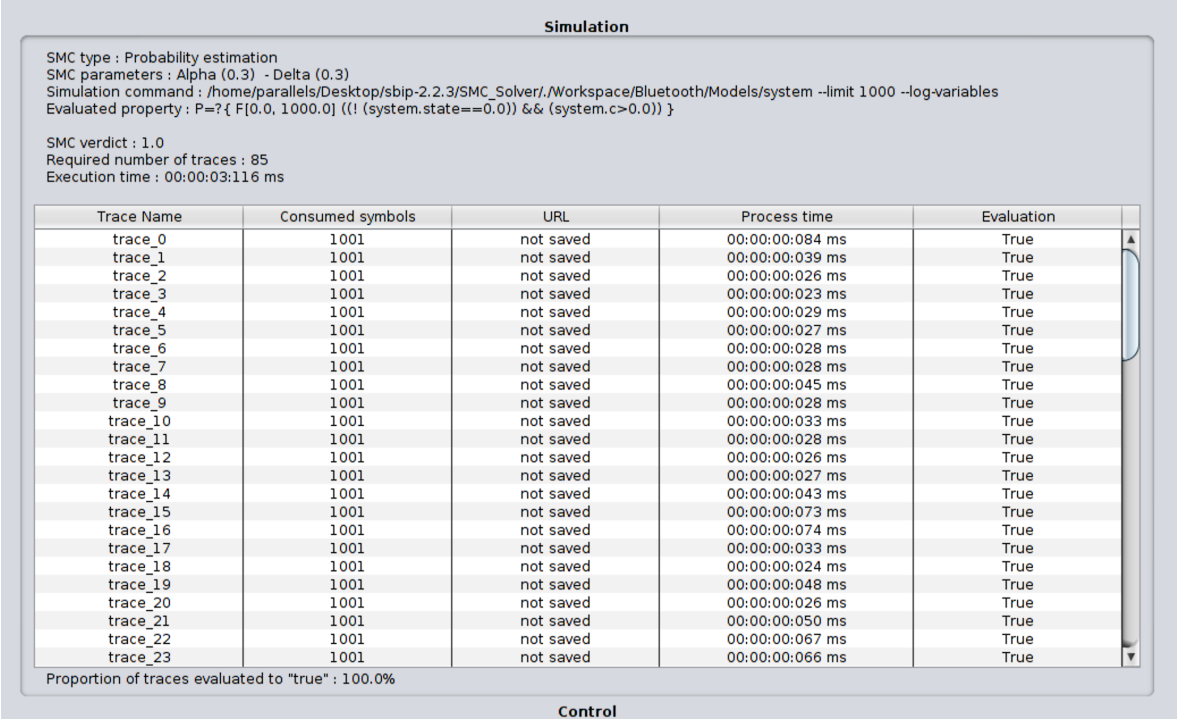


Рисунок 18 – Результаты исследования ограничения по формуле (2)

Результаты проверки ограничений по формуле (3) представлены на рисунке 19.

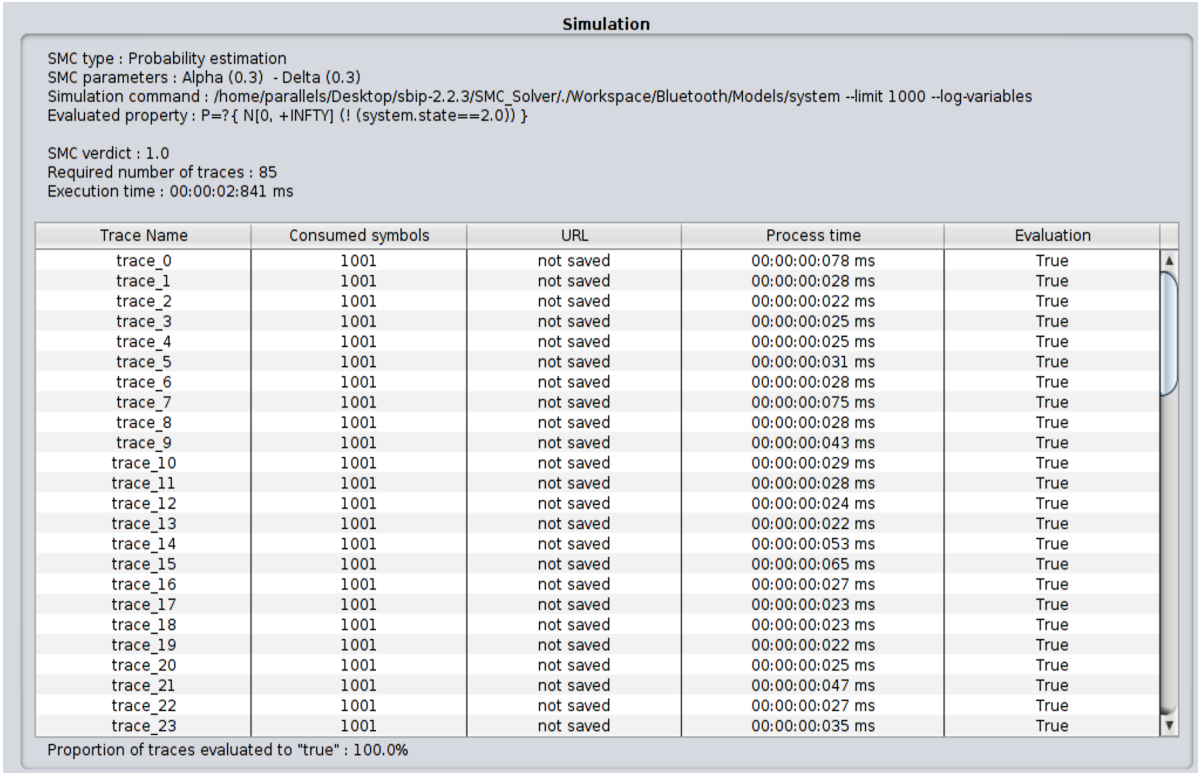


Рисунок 19 – Результаты исследования ограничения по формуле (3)

Результаты проверки ограничений по формуле (4) представлены на рисунке 20.

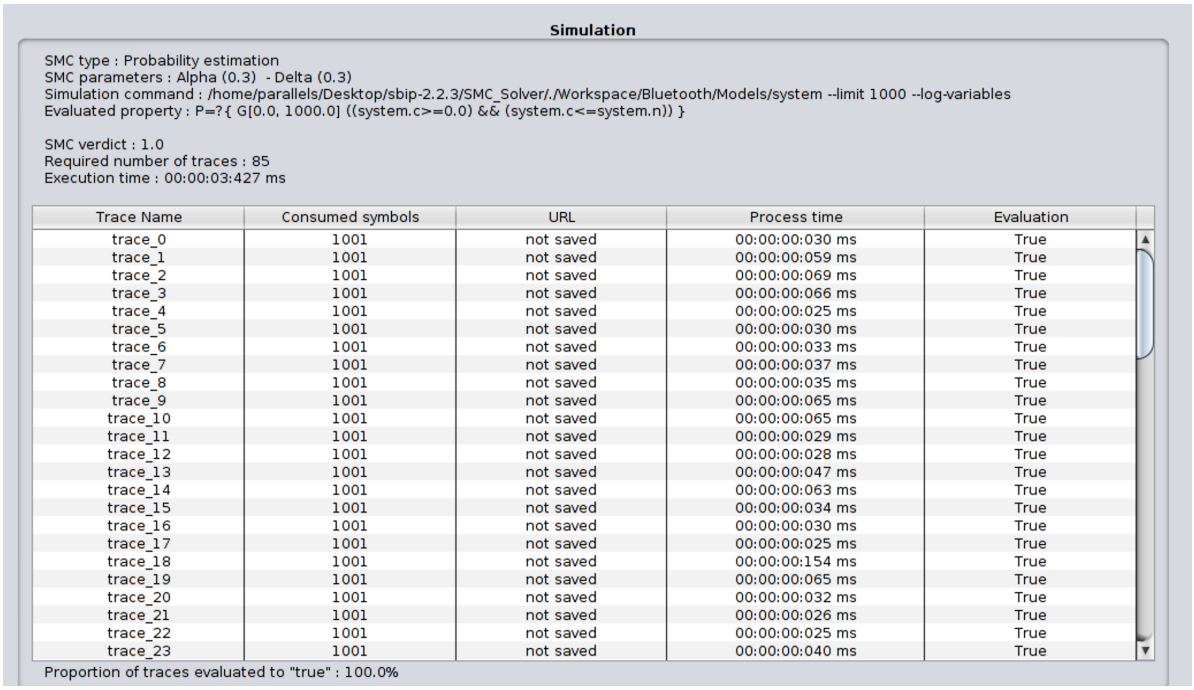
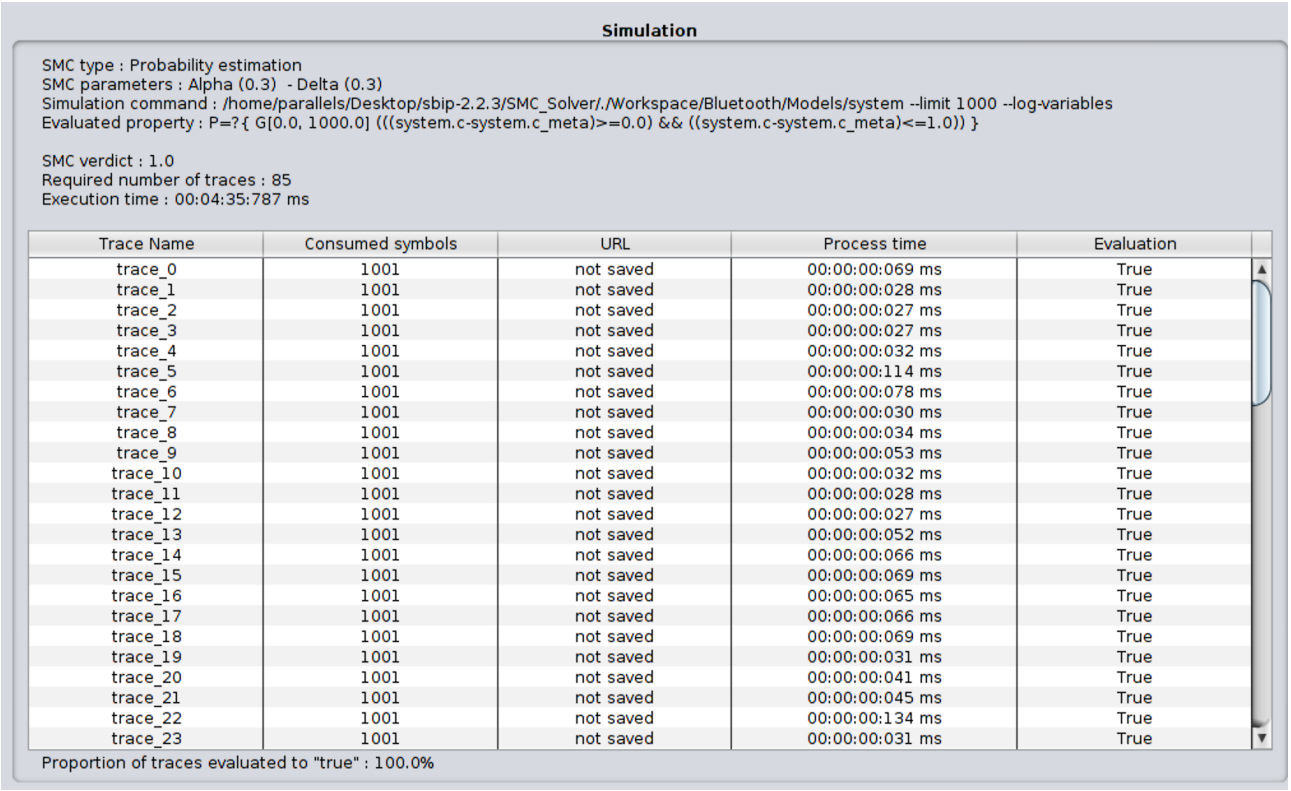


Рисунок 20 – Результаты исследования ограничения по формуле (4)

Результаты проверки ограничений по формуле (5) представлены на рисунке 21.



Simulation

SMC type : Probability estimation
SMC parameters : Alpha (0.3) - Delta (0.3)
Simulation command : /home/parallels/Desktop/sbip-2.2.3/SMC_Solver/./Workspace/Bluetooth/Models/system --limit 1000 --log-variables
Evaluated property : P=?{ G[0.0, 1000.0] (((system.c-system.c_meta)>=0.0) && ((system.c-system.c_meta)<=1.0)) }

SMC verdict : 1.0
Required number of traces : 85
Execution time : 00:04:35:787 ms

Trace Name	Consumed symbols	URL	Process time	Evaluation
trace_0	1001	not saved	00:00:00:069 ms	True
trace_1	1001	not saved	00:00:00:028 ms	True
trace_2	1001	not saved	00:00:00:027 ms	True
trace_3	1001	not saved	00:00:00:027 ms	True
trace_4	1001	not saved	00:00:00:032 ms	True
trace_5	1001	not saved	00:00:00:114 ms	True
trace_6	1001	not saved	00:00:00:078 ms	True
trace_7	1001	not saved	00:00:00:030 ms	True
trace_8	1001	not saved	00:00:00:034 ms	True
trace_9	1001	not saved	00:00:00:053 ms	True
trace_10	1001	not saved	00:00:00:032 ms	True
trace_11	1001	not saved	00:00:00:028 ms	True
trace_12	1001	not saved	00:00:00:027 ms	True
trace_13	1001	not saved	00:00:00:052 ms	True
trace_14	1001	not saved	00:00:00:066 ms	True
trace_15	1001	not saved	00:00:00:069 ms	True
trace_16	1001	not saved	00:00:00:065 ms	True
trace_17	1001	not saved	00:00:00:066 ms	True
trace_18	1001	not saved	00:00:00:069 ms	True
trace_19	1001	not saved	00:00:00:031 ms	True
trace_20	1001	not saved	00:00:00:041 ms	True
trace_21	1001	not saved	00:00:00:045 ms	True
trace_22	1001	not saved	00:00:00:134 ms	True
trace_23	1001	not saved	00:00:00:031 ms	True

Proportion of traces evaluated to "true" : 100.0%

Рисунок 21 – Результаты исследования ограничения по формуле (5)

По результатам проверки адекватности модели можно сделать вывод что модель работает корректно и поведение системы остается в рамках заданных ограничений.

Также было проведено параметрическое исследование по формуле (6) результаты были представлены на рисунке 22.

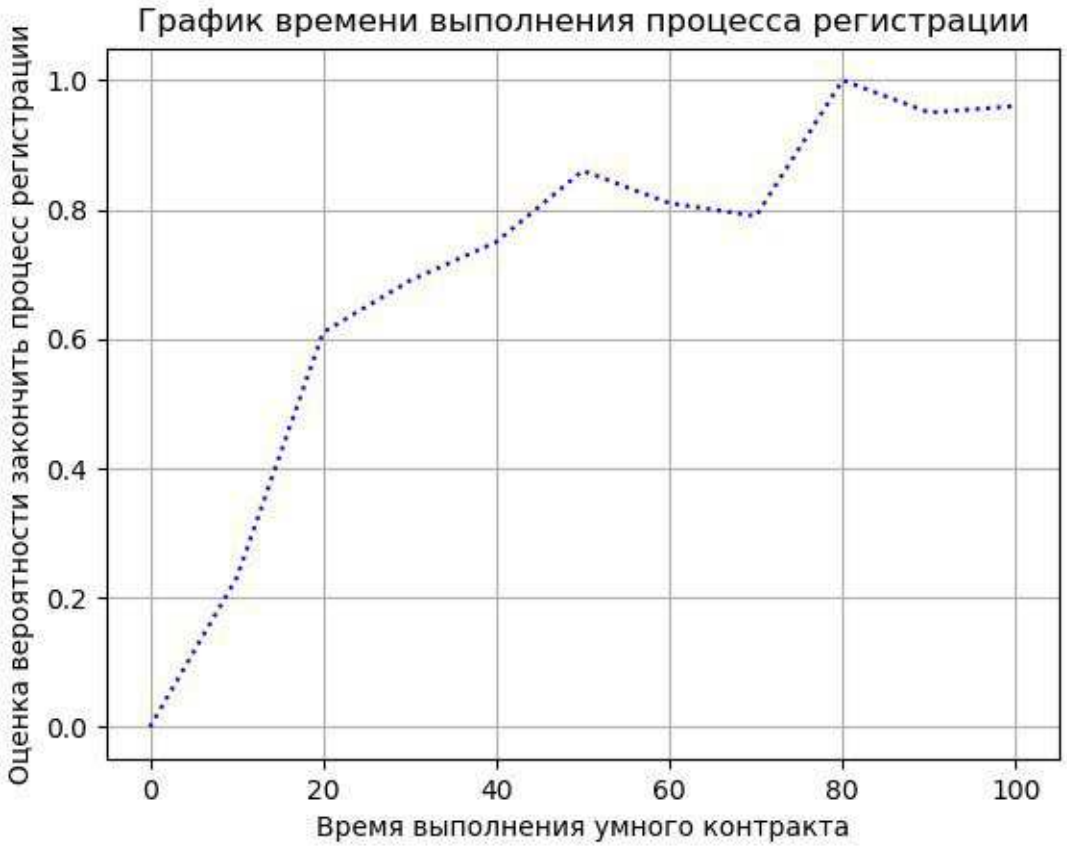


Рисунок 22 – Результаты параметрического исследования по формуле (6)

На основании данного параметрического исследования можно сделать вывод, что оценка вероятности завершить процесс регистрации пользователя ощутимо зависит от времени выполнения в рамках 0 до 100 мс – чем больше времени проходит регистрация, тем больше вероятность получить псевдоним.

ЗАКЛЮЧЕНИЕ

При рассмотрении приложений, работающих в среде распределенных реестров и случаев взлома подобных приложений было выявлено, что основная причина наличия уязвимостей в программном обеспечении, является недостаточное тестирование программного обеспечения. Для того чтобы повысить качество разрабатываемого программного обеспечения, были проведено исследование возможности применения методов формальной верификации для приложений, работающих в среде распределенных реестров.

На основе анализа существующих программных решений для проведения формальной верификации была выбрана система BIP Framework исходя из следующих критериев:

- возможность моделирования стохастической системы;
- возможность проведения параметрического исследования;
- формирование графиков сравнительного анализа;
- подключение пользовательского исполняемого кода.

В качестве исследуемой модели был выбрана модель системы, которая выдает пользователю псевдоним.

На основе требований из первого раздела и составленной модели и ограничений из второго раздела, была решена задача по проведению формальной верификации системы. Она состояла из следующих этапов:

- создание модели;
- формирование ограничений;
- симуляция работы системы и проверка работы системы в рамках заданных ограничений.

В ходе исследования была выдвинута и экспериментально проверена гипотеза о том, что возможно построение и формальное исследование модели умного контракта для регистрации новых пользователей. Была построена модель, для которой на языке линейной темпоральной логики сформулировано

ограничение на состояние модели, отражающее особенности процесса регистрации в условиях нехватки ресурсов. Формальная верификация построенной модели позволила оценить степень достоверности сформулированного ограничения.

Результаты исследования прошли апробацию на международной научно-практической конференции и опубликованы в сборнике трудов [25].

В дальнейшем, планируется расширить модель исследования и добавить в исследуемую модель злоумышленника, чтобы проанализировать возможность получения чужого пользовательского псевдонима в процессе регистрации. Также предполагается использовать метод проверки гипотез для анализа данных полученных при исследовании.

СПИСОК СОКРАЩЕНИЙ

DOS - denial of service;

DSL - domain specific language;

ЭЦП - электронная цифровая подпись;

NASA - National aeronautics and space administration;

PRISM - probabilistic model checker;

PCTL - probabilistic computation tree logic;

CSL - computation tree logic;

LTL - linear temporal logic;

BIP - behavior interaction priority;

PESTIM - probability estimation;

GUI - graphical user interface;

SSE - stochastic simulation engine;

MM - monitoring module;

SMCE - statistical model checking engine;

PEM - parametric exploration module.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Программа «Цифровая экономика Российской Федерации» : распоряжение правительства Российской Федерации от 28 июля 2017 г. №1632-р // Председатель правительства Российской Федерации. с. 1-5.

2 Шадрин, Ф. Г. Технология blockchain. Смарт-контракты / Ф. Г. Шадрин // Электронный сборник статей по материалам XLIV студенческой международной научно-практической конференции / Университетская книга. – Курск. – 2018. – С. 110–117.

3 Stuart Haber. How to TimeStamp a Digital Document / Stuart Haber, W Scott Stornetta // Journal of Cryptology. – 1991. – Vol.3, No. 2, P. 99–111.

4 Lotte Fekkes. Comparing Bitcoin and Ethereum : Bachelor thesis Computer Science : 0613 / Lotte Fekkes. – Nijmegen, 2018. – P. 43.

5 Matteo Gianpietro Zago. 50+ Examples of How Blockchains are Taking Over the World [Электронный ресурс] // Medium. – San Francisco, 2018. – Режим доступа: https://medium.com/@matteozago/50-examples-of-how-blockchai ns-are-taking-over-the-world-4276bf488a4b.

6 Ходаков, А. Д. Особенности разработки смарт-контракта на базе блокчейн платформы ethereum / А. Д. Ходаков, С. А. Зайкова // Сборник научных трудов XIII Международной научно-практической конференции / Ассоциация научных сотрудников «Сибирская академическая книга». – Новосибирск. – 2018. – С. 205–207.

7 Karthikeyan Bhargavan. Formal Verification of Smart Contracts / Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Anitha Gollamudi // Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security. – 2016. – P. 91–96.

8 Romain Edelmann. Behaviour-Interaction-Priority in Functional Programming Languages: Formalisation and Implementation of Concurrency Frameworks in Haskell

and Scala / Romain Edelmann, Simon Bliudze, Joseph Sifakis // Infoscience. EPFL scientific publications. – 2015. – P. 201.

9 Jeff Hu. Learn Blockchain's Top 25 Hacks in History [Электронный ресурс] // Medium. – San Francisco, 2019. – Режим доступа: https://hackernoon.com/tech-explained-top-24-blockchain-hacks-in-history-first-half-40c390dc4a96.

10 Сергеев, Д. В. Верификация и способы описания формальных моделей / Д. В. Сергеев // Альманах современной науки и образования / Издательство «Грамота». – Тамбов. – 2011. – № 4. – С. 84–85.

11 Fuzhi Wang. Symbolic implementation of model-checking probabilistic timed automata : Thesis for the degree of doctor of philosophy : 0223 / Fuzhi Wang. – Birmingham, 2006. – P. 166.

12 Robert C. Armstrong. Survey of Existing Tools for Formal Verification / Robert C. Armstrong, Ratish J. Punnoose, Matthew H. Wong, Jackson R. Mayo // Center for Computing Research. – 2014. – P. 40.

13 Christel Baier. Principles of Model Checking / Christel Baier, Joost-Pieter Katoen. – .: The MIT Press, 2018. – P. 984.

14 Stacy D. Nelson. Formal Verification for a Next-Generation Space Shuttle / Stacy D. Nelson, Charles Pecheur. // Formal Approaches to Agent-Based Systems, 2002. - P. 53-57.

15 Jasim O. Formal verification of quadcopter flight envelop using theorem prover / Jasim O. ,Veres S.M. // Proceedings of 2018 IEEE Conference on Control Technology and Applications (CCTA), 2018. - P. 1502-1507

16 Marta Kwiatkowska. PRISM 4.0: Verification of Probabilistic Real-time Systems / Marta Kwiatkowska, Gethin Norman, David Parker // In Proc. 23rd International Conference on Computer Aided Verification (CAV'11). – 2011. – Vol. 6806, P. 585–591.

17 Баринова, А. А. Методы и средства обеспечения конфиденциальности смарт-контрактов / А. А. Баринова, С. В. Запечников // Безопасность

информационных технологий / – КлАССное снаряжение. – Москва. – Т. 24, № 2. – С. 16–23.

18 Isabelle [Электронный ресурс] : Официальный сайт программного обеспечения для верификации Isabelle - Режим доступа : https://www.prismmodelchecker.org/

19 K-tool [Электронный ресурс] : Официальный сайт программного обеспечения для верификации K-tool - Режим доступа : http://www.kframework.org/index.php/Main_Page/

20 BIP Framework [Электронный ресурс] : Официальный сайт программного обеспечения для верификации BIP Framework - Режим доступа : http://www-verimag.imag.fr/New-BIP-tools.html?lang=en

21 Tesnim Abdellatif. Formal verification of smart contracts based on users and blockchain behaviors models / Tesnim Abdellatif, Kei-Léo Brousmiche // IFIP NTMS International Workshop on Blockchains and Smart Contracts (BSC). – 2018. – P. 5.

22 Braham Lotfi Mediouni. SBIP 2.0: Statistical Model Checking Stochastic Real-time Systems / Braham Lotfi Mediouni, Ayoub Nouri, Marius Bozga, Mahieddine Dellabani // Verimag Research Report № TR-2018-5. – 2018. P. 22.

23 Jérôme Darbon. Approximate Probabilistic Model Checking for Programs / Jérôme Darbon, Richard Lassaigne, Sylvain Peyronnet // LRDE/EPITA. – 2007. P. 6.

24 Wassily Hoeffding. Probability inequalities for sums of bounded random variables / Wassily Hoeffding // Department of Computer Science and Electrical Engineering. – 2008. – P. 13 – 29.

25 Максимов, Д. Б. Использование методов формальной верификации для оценки качества умных контрактов / Д. Б. Максимов, И. А. Якимов, А. С. Кузнецов // ИНТЕЛЛЕКТУАЛЬНЫЙ ПОТЕНЦИАЛ ОБЩЕСТВА КАК ДРАЙВЕР ИННОВАЦИОННОГО РАЗВИТИЯ НАУКИ. - Тюмень. - 2019 - Часть 1 С. 57 - 62.

26 Kristin Y. Rozier. Linear Temporal Logic Symbolic Model Checking // Computer Science Review, 2011. P. 163-203

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт космических и информационных технологий
Кафедра «Информатика»

УТВЕРЖДАЮ
Заведующий кафедрой
_____ Кузнецов А. С.
«10» 07 2019 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Алгоритмическое и программное обеспечение оценки качества приложений, работающих в среде распределенных реестров

09.04.04 Программная инженерия
09.04.04.01 Программное обеспечение вычислительной техники и автоматизированных систем

Научный руководитель	09.07.19	доцент, канд. техн. наук	А. С. Кузнецов
Выпускник	09.07.19		Д. Б. Максимов
Рецензент	09.07.19	доцент, канд. техн. наук	Д. В. Капулин

Красноярск 2019